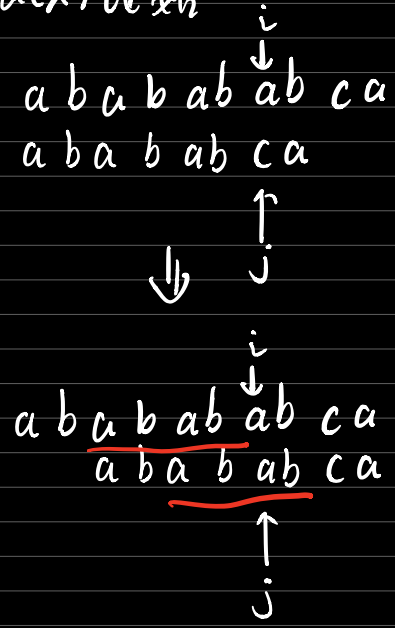
KMP算法

KMP算法是为了加速字符串匹配的算法，其有一核心为next数组

index	0	1	2	3	4	5	6	7
模式串：	a	b	a	b	a	b	c	a
next	-1	0	0	1	2	3	4	0

先谈 next 数组含义：next[j] 表示第 j 位匹配失败后 jump 到的 index，比如

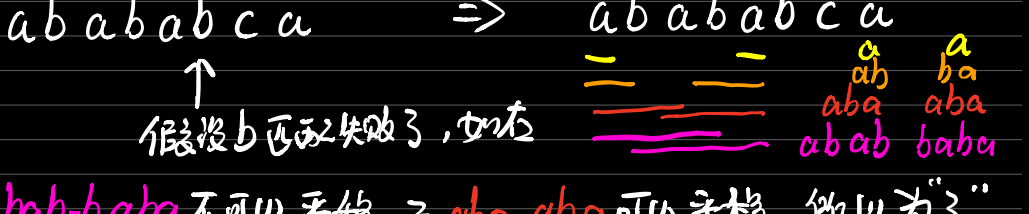


j 和 i 无法匹配上了 j = next[j]

因为红色这两段一样，所以可以平移过去不用再匹配对.

如上我们可以知，其实除 0 就是前后缀相同的最大长度

"字符串是从左往右读的"，起招上面规律，我们可以手算 next



显然 abab-baba 不可以平移，而 aba-aba 可以平移，所以为了"当然 a 也是可以的不过因为可以选择更长的了，就不使用了.

注意 next 为 0 或

快速生成next的方法：

其实其思路就是自己和自己匹对.

next[0]=-1 恒定.

next[1]=0

a b a b a b c a

a b a b a b c a

next[2]=0

a b a b a b c a

a b a b a b c a

next[3]=1

a b a b a b c a

a b a b a b c a

next[4]=2

next[5]=3

next[6]=4

a b a b a b c a

a b a b a b c a

为什么不用移动，因为已经匹配了，能匹配多长

a b a b a b c a

next[7]=0

没能匹配对 c

到这就已经结束了. 总的算法

begin j=1:

next[0]=-1

next[1]=0

无法匹对 j=0

a b a b a b c a

a b a b a b c a

next[i]=0 无法匹对

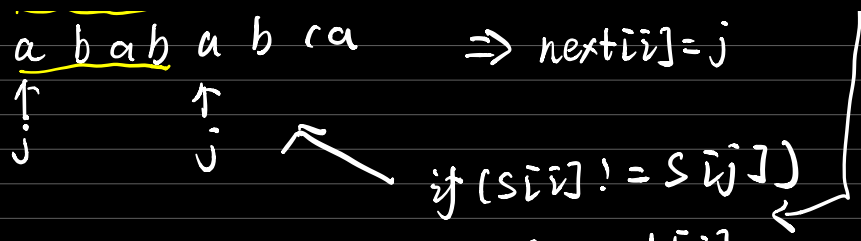
if(s[i]!=s[j])

⇒ next[i]=0 (j=0)

i=2

a b a b a b c a

if(s[i]==s[j])



按照上面的思想递规，可知KMP的next数组的代码如下(Java/C++)

```
int  i=1, j=0;
int[] next =new int[s.length];
next[0]=-1; next[1]=0;
while(i<s.length()){
    if(s.charAt(i)==s.charAt(j)){
        next[i]=j;
        i++;
        j++;
    }else{
        j=next[j];
    }
}
```

j=-1时同样是往下找，不然会报错，找到-1意味着无法匹配

```
||(j==-1)
```